Chat für Schwerhörige

Eine Übung zur Realisierung eines Programms mit dem Decorator-Pattern

10.12.2014

Melanie Göbel, Patricia Kronowetter

Inhalt

Aufgabenstellung

Aufgabe für 2 Personen (Gruppeneinteilung siehe unten)
Erstellt ein einfaches Chat-Programm für "Schwerhörige", mit dem Texte zwischen zwei Computern geschickt werden können.

Dabei soll jeder gesendete Text "geschrien" ankommen (d.h. ausschließlich in Großbuchstaben, lächeln wird zu *lol*, Buchstaben werden verdoppelt, … - ihr dürft da kreativ sein)

Zusätzlich sollen "böse" Wörter ausgefiltert und durch "$%&*" ersetzt werden. Diese Funktionalität soll aber in der GUI jederzeit aktiviert und deaktiviert werden können. Die "bösen" Wörter sollen in einem Properties-File definiert werden können.

Verwendet dafür ausgiebig das Decorator-Pattern.

Nähere Informationen zum Transport von Daten über das Netzwerk findet ihr hier[1]

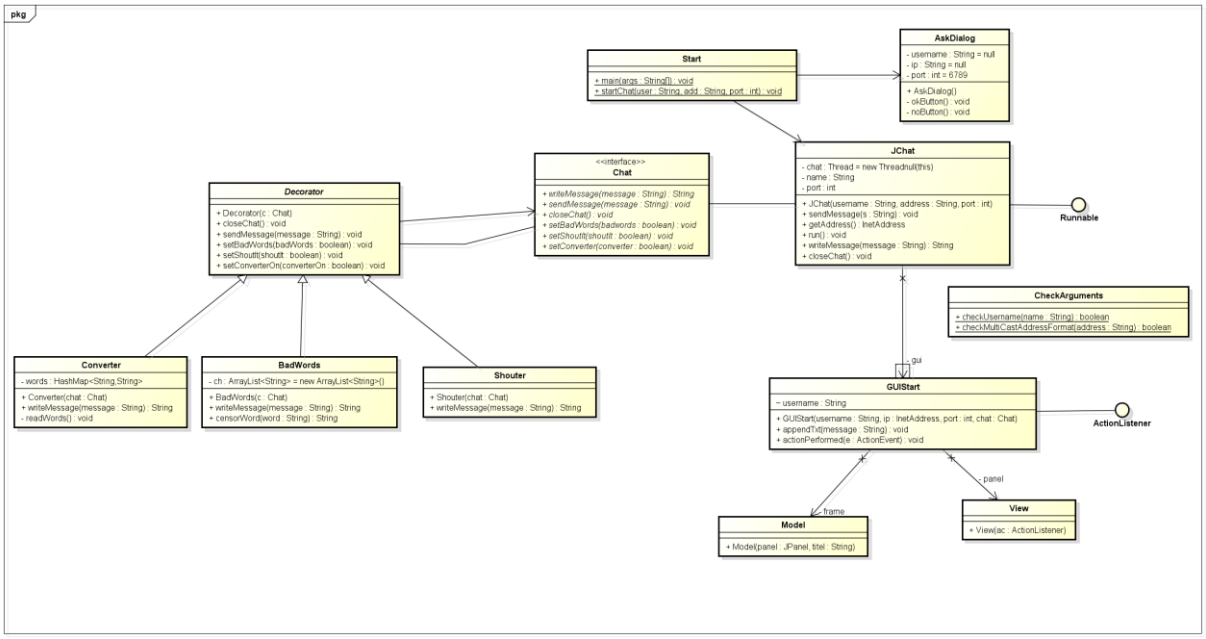
Zeitaufteilung

Die Arbeitsaufteilung beinhaltet die Aufteilung der Arbeit zu den Personen, die geschätzte Arbeitszeit sowie die endgültig, benötigte Zeit.

Arbeit	Zuständig	Geschätze Zeit (in min)	Benötigte Zeit (in min)
Networking – Senden Empfangen	Göbel	150	160
GUI – Grundgerüst	Göbel	15	15
GUI fertigstellen	Kronowetter	40	10
Dekorationen	Kronowetter	60	10
Testing	Göbel, Kronowetter	30,30	30,0
Dokumentation	Göbel, Kronowetter	30,30	60,0
Zusätzliche Arbeit	Göbel	0	100
Zusätzliche Arbeit	Kronowetter	0	0
Insgesamt	Göbel, Kronowetter	385	385
	Göbel	215	365
	Kronowetter	160	20

Design

Für das Realisieren der Dekorationen wird ein Decorator-Pattern angewendet. Für die graphische Oberfläche wurde das MVC-Pattern angewendet. Die Start Klasse ruft AskDialog und JChat auf, dort ist die main-Methode.



Dokumentation der Funktionalitäten

Multicast-Chat

Das Chatsystem wurde inspiriert von einem Beispiel [2], dass eine einfach Applikation für einen Chat erklärt. Für die Kommunikation wird ein Multicastsocket verwendet dieser Socket benötigt einen Port auf dem die Nachrichten gesendet werden und eine Multicastadresse (IP Adresse Klasse D) die zwischen 224.0.0.0 und 239.255.255.255 ist [3]. Nachrichten werden gesendet mit sendMess(String message). Nachrichten werden empfangen und mit writeMessage geschrieben, wobei die Nachricht mit einen \n versehen werden. Dem Multicast-Socket werden Nachrichten übermittelt indem man sie in einem Datagrampaket-Objekt gespeichert werden.

Check Arguments

Die Klasse Check-Arguments.java wurde von der Aufgabe in Dezentrale Systeme wiederverwendet und für die Aufgabenstellung erneuert. Es beinhaltet 2 Methoden, die mit regulären Ausdrücken Username und IP-Adresse überprüfen. Die Methode für die IP-Adresse wurde so umgeändert, dass es nur überprüft ob es eine IP der Klasse D ist.

Username
```
^[a-z A-Z 0-9_-]{3,15}$
```

Multicast IP-Adresse
```
2(?:2[4-9]|3\\d)(?:\\.(?:25[0-5]|2[0-4]\\d|1\\d\\d|[1-9]\\d?|0)){3}
```

AskDialog

Ist ein kleines JDialog Fenster zum Erfragen der Paramtern. Username, MulticastIP-Address und Port werden überprüft.

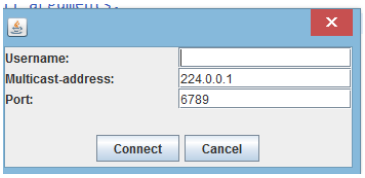


Abb1: JDialog wenn man es startet

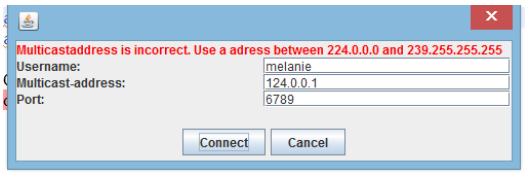


Abb2: JDialog mit Fehlermeldung

GUI

Die GUI besteht aus den Klassen GUIStart.java, Model.java, View.java. In einem ScrollPane ist eine TextArea, diese zeigt den gesendeten Text. Im unteren Bereich des Hauptpanels mit BorderLayout ist ein Panel mit BoxLayout Textfeld und ein Sende-Button. Im oberen Bereich des BorderLayouts ist ein Panel mit mehreren Buttons. Wenn Enter gedrückt wird, sendet GUIStart.java den Text vom Textfeld weiter (ebenso wie beim Sende-Button).

GUIStart.java = Startet die GUI und kümmert sich um die Aktionen von View (ActionListener)
View: Zeit Panels an, die die GUI darstellen (Textfeld, Button, etc)
Model: Erstellt ein Fenster mit den übergebenen Panel (View.java)

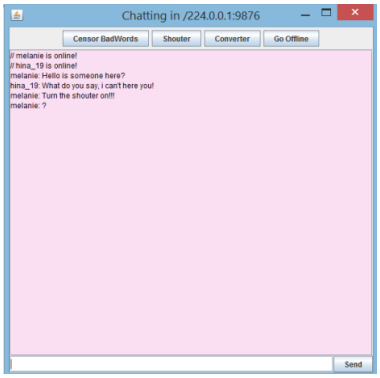


Abb3: GUI für den Chat

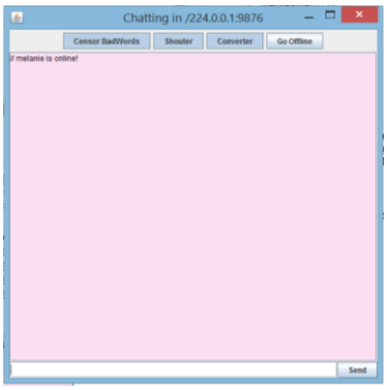


Abb4: GUI für den Chat mit allen Decorator an

Decorator

Als Dekoration haben wir 3 Klassen die die abstrakte Klasse Decorator erben. Alle Dekoration implementieren die Methode writeMessage(). Aufgerufen kann eine Dekoration wie ein Chat (da die Abstrakte Klasse Decorator von Chat erbt, siehe Design). Um nicht alle Methoden implementieren zu müssen wird im Konstruktor von Decorator ein Chat übergeben und in ein Attribut gespeichert (dies ist die Funktionsweise vom Decorator-Pattern).

Shouter
Schreibt die Nachricht sowie den User von dem die Nachricht ist groß, in der Umgangssprache wird das schreien bezeichnet da großgeschriebene Wörter geschrien werden. Die Funktionalität geruht auf einen einfach .toUpperCase() auf die Nachricht.

Converter
Ändert bestimmte Wörter zu anderen um. Diese Änderungen können in ein File eingespeist werden. Dort steht zum Beispiel: Mädchen;Mädl
Das bedeutet, dass das Wort Mädchen zu Mädl umgeschrieben wird.

BadWords
Zensiert alle Schimpfwörter die in einen File drinnen stehen. Zensiert werden sie mit Sonderzeichen ($%&*§). Alle Schimpfwörter die in den File drinnen stehen sind nicht Case-Sensitive für die Suche in der Nachricht (mit toUpperCase wird gesucht). Für jedes Wort der Nachricht wird im File nachgesehen ob es dort vorhanden ist, wenn ja soll ein „Wort“ aus Sonderzeichen mit derselben Zeichenanzahl stattdessen geschrieben werden.

Probleme und Lösungen

Probleme von Melanie Göbel

Die TextArea für den gesendeten Text war schreibbar. Lösung: setEditable(false) setzen, somit sind keine Nachrichten fälschbar in der eigenen GUI.

Manchmal sendet der Chat jede Nachricht doppelt.

Die Funktionalität von BadWords und Converter ist nicht für alle Fälle immer anwendbar (zb nicht wenn Shouter davor an ist). Diese Funktionalität ist unkommentiert und unübersichtlich vom Aufgabenpartner. Lösung: Dokumentieren und Sicherstellen ob Aufgabenpartner dazu fähig ist.

Sende-Button und Textfeld gleich groß, obwohl es gewünscht ist dass der Button kleingenug ist Lösung: Anderes Layout verwenden, BoxLayout.

Das Arbeiten mit nicht dokumentiertem Code ist schwierig, wenn der Code auch noch nicht von sich selbst ist. Auch wenn der Code funktioniert, kann er nicht wirklich verwendet werden. Ebenso werden Fehler erst mit Einarbeit gefunden. Lösung: Stets Dokumentation verlangen, sonst nichts annehmen, die Zeit es zu verstehen kostet viel zu viel!

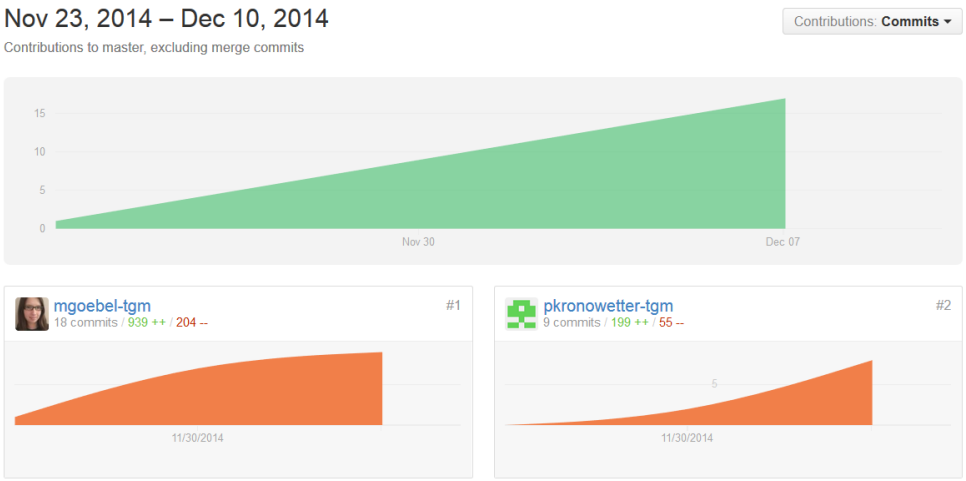
Probleme von Patricia Kronowetter

Falls Probleme vorhanden, sind sie hier eingetragen.

Genaue Arbeitsaufteilung

Github

In dieser Graphik sieht man die Trends der Commits und hinzugefügte/gelöschte Zeilen. Insgesamt gibt es 548 hinzugefügte und 69 gelöschte Zeilen.



Commit log ist unter doc/Github.log vorhanden.

Autoren mit Verhältnis der Arbeit der Klassen

Klassenname	Verhältnis Göbel:Kronowetter
JChat.java	1:0
AskDialog.java	1:0
BadWords.java	1:1
Converter.java	1:1
Shouter.java	0:1
CheckArguments.java	1:0
GUIStart.java	1:0
View.java	1:0
Model.java	1:0
Decorator.java	0:1
Chat.java	2:1
Decorator.java	1:1

Quellen

[1] “Networking Basics” – Oracle
http://docs.oracle.com/javase/tutorial/networking/overview/networking.html
(zuletzt geöffnet: 2.12.2014)

[2] „How to create Chat Application in Java“- Vinay Kumar
http://mrbool.com/how-to-create-chat-application-in-java/26778
(zuletzt geöffnet: 2.12.2014)

[3] Class MulticastSocket- Oracle
http://download.java.net/jdk7/archive/b123/docs/api/java/net/MulticastSocket.html
(zuletzt geöffnet 3.12.2014)